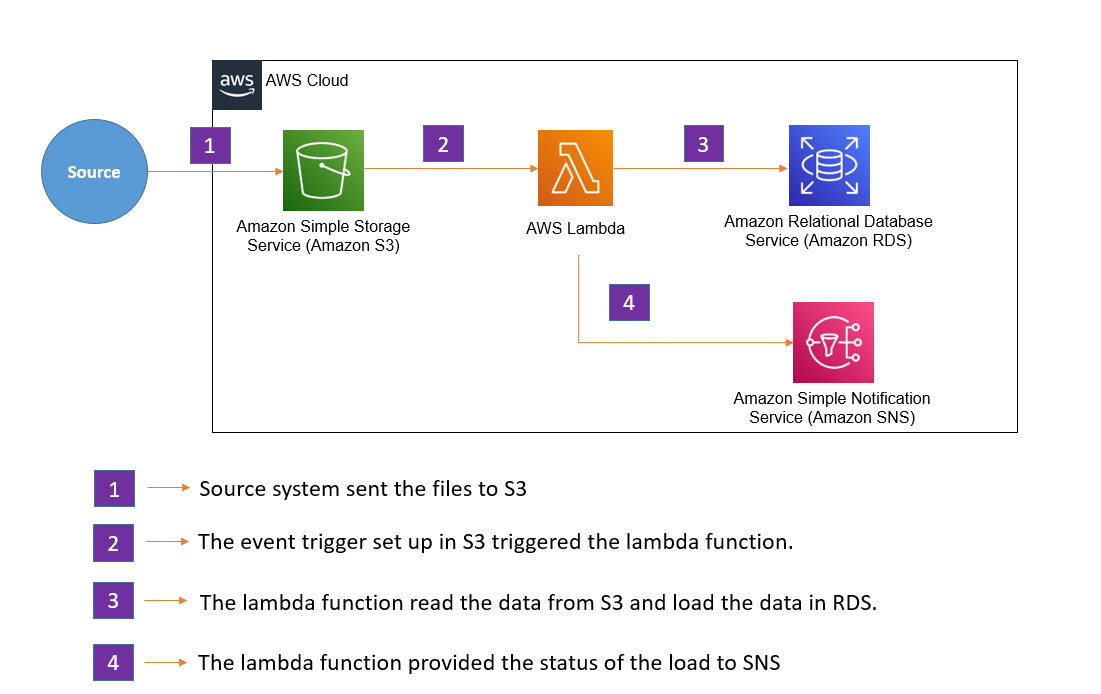
Data Engineering – Use Case – Read the file from S3 and Load it in RDS.

Problem Statement - An organization received source systems data to an input landing. The requirement is to load the data to a relational database system so that BI reports can be developed. The average number of records in the file is 10000.

Solution: Looking at the problem statement and file size, we are going to develop an event driven architecture to load the data. The solution steps.

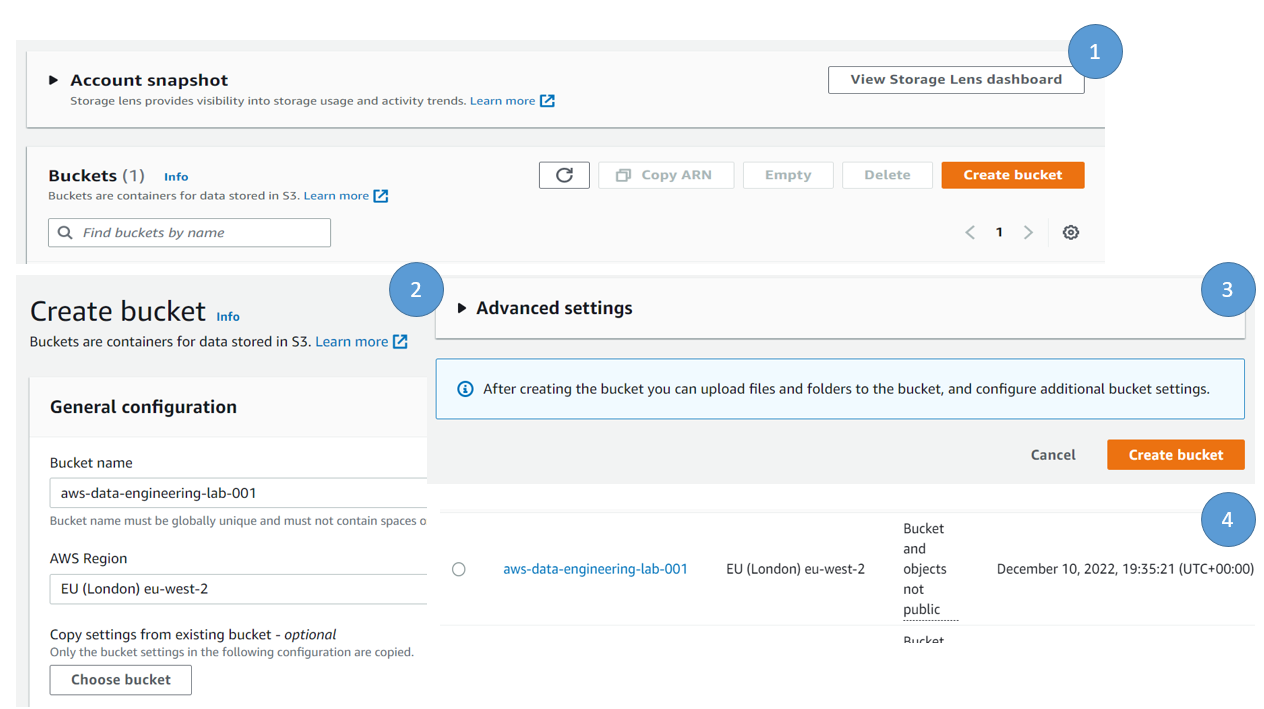
1. The input files are placed in S3 bucket.
2. An event is triggered, which in turn
3. The lambda function extracts the file details from event context and load the data in RDS.

We are going to follow below solution architecture to solve this problem.

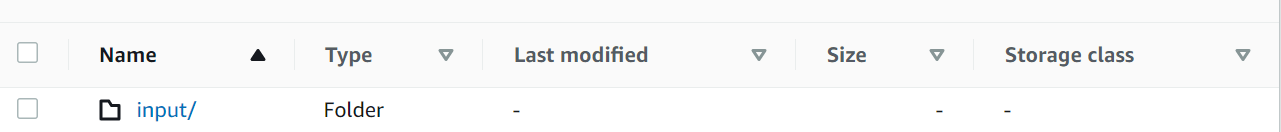


Let’s start’s the e2e action on this.

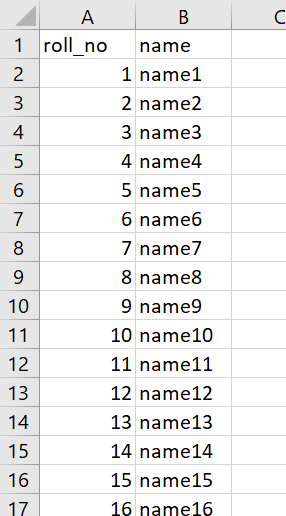
Step -1: Create a bucket called ***aws-data-engineering-lab-001*** for our data engineering project.



Step -2: Create a folder called ***input*** where we will store the input files.



Step -3: For this demo, we created a demo file which is having 2 fields ( roll\_no, name). The sample record of the file is



Step -4: For Event trigger set up with S3, first create a lambda function.

1. Search lambda in aws console search and select lambda and click create function under lambda function.
2. Provide the Basic information. Here we selected the run time for lambda function is python3.9
3. Under permission, attach a role to lambda which have permission to S3, RDS and basic execution role of lambda. Here we attached a role which have the require permissions to S3, RDS so that lambda function can interact with require aws services. Then create function.

Note – It is better to create an IAM role in advance so that user can select the role from drop down menu under lambda function.

1. Copy the below code to lambda console – Code source. We are going to provide details on the code later part of the blog.

import json

import boto3

import os

import mysql.connector

import sys

import logging

import csv

logger = logging.getLogger()

logger.setLevel(logging.INFO)

#Connection String Details - Define as part of environmental variable details.

host = os.environ["host"]

port = os.environ["port"]

database = os.environ["database"]

username = os.environ["username"]

password = os.environ["password"]

def lambda\_handler(event, context):

    bucket\_name = event['Records'][0]['s3']['bucket']['name']

    file\_key\_name = event['Records'][0]['s3']['object']['key']

    s3\_client = boto3.client('s3')

    s3\_file\_path = bucket\_name + '/' + file\_key\_name

    try:

       # connection = create\_mysql\_DBconnection()

        connection = mysql.connector.connect(user=username,password=password,host=host,port=port,database=database)

        file\_object = s3\_client.get\_object(Bucket=bucket\_name,Key=file\_key\_name)

        record\_lines = file\_object['Body'].read().decode('utf-8').split()

        records = []

        for line in csv.DictReader(record\_lines):

            record\_value =list(line.values())

            records.append(record\_value)

        insert\_statement="insert into demo\_data (roll\_no,name) values (%s,%s);"

        cs = connection.cursor()

        print("Executing....")

        cs.executemany(insert\_statement,records)

        cs.close()

        connection.commit()

    except Exception as e:

        logger.info("###### Database Exception Occurred ######\*")

        logger.error(str(e))

    return {

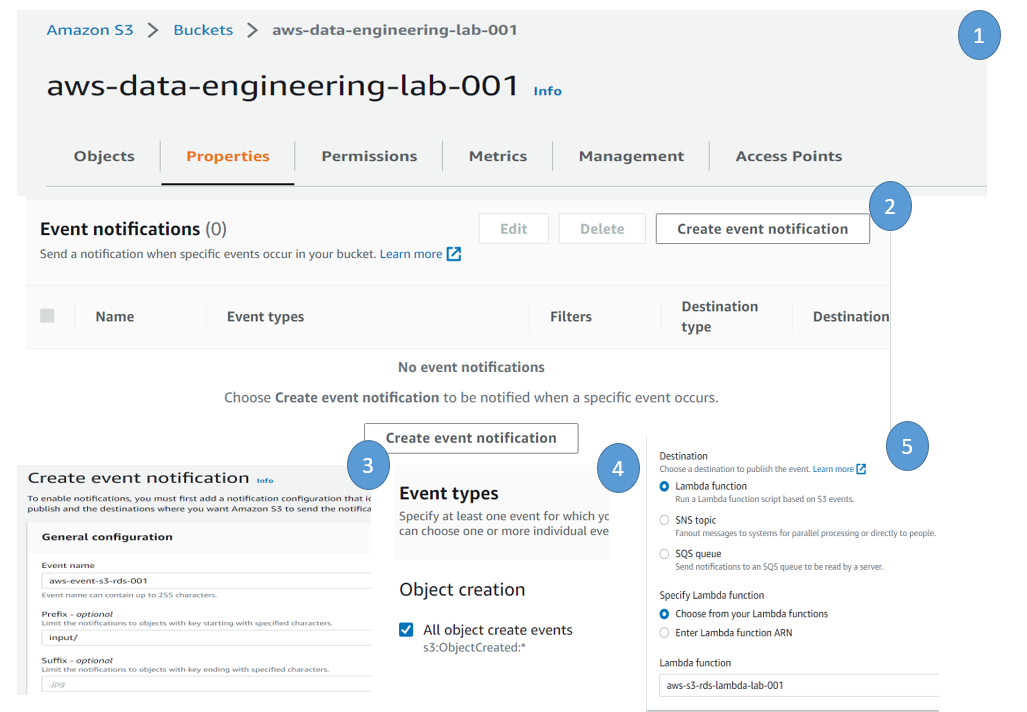
        'statusCode': 200,

        'body': json.dumps('Hello from Lambda!')

    }

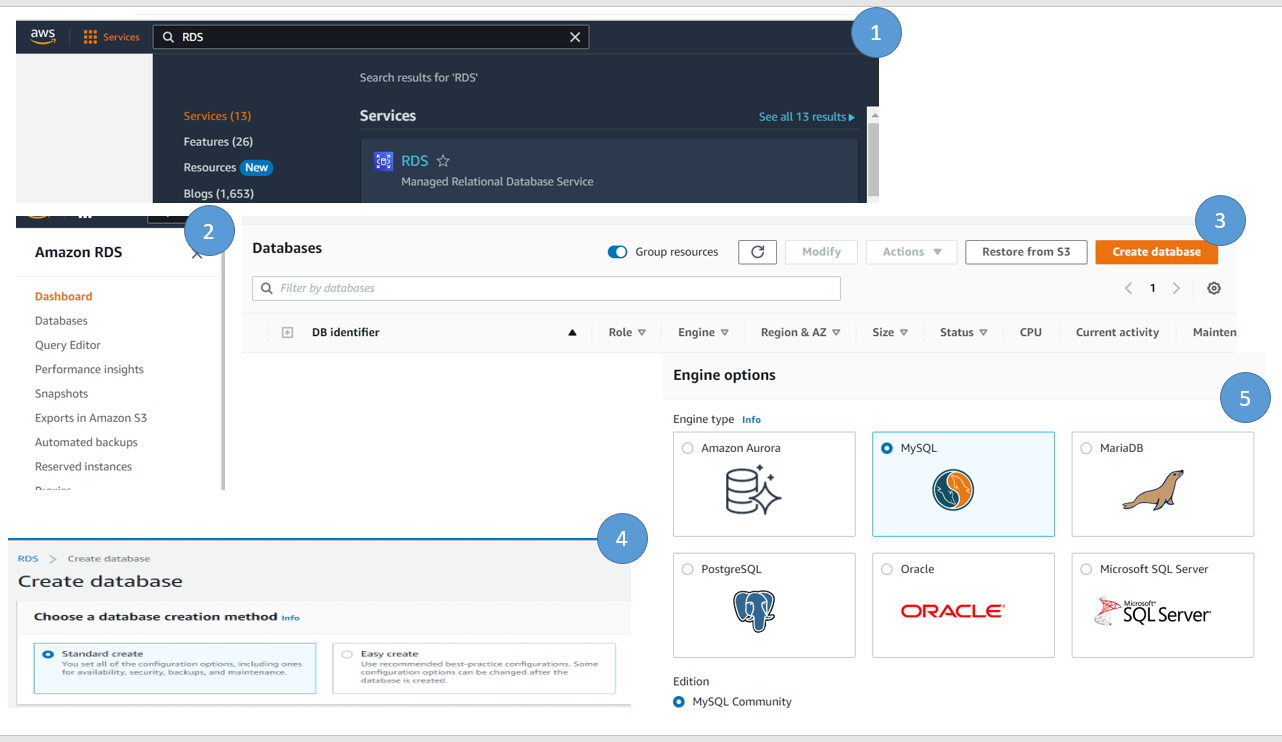
Step -5: To enable the event trigger in S3 (when the file is upload in the S3), it will call the lambda function.

1. Go to S3 bucket - aws-data-engineering-lab-001 and go to the properties tab.
2. Under Event notification, click create event notification.
3. Provide details on General configuration like name of the event, in the prefix (provide the folder name, in our case it is input).
4. On Event type, for demo purpose we selected the all objects creation. The remaining options we can kept as it but for as per business requirement and detailing, one can select the appropriate options as required.
5. Under destination, select lambda function and select the right lambda function and finally save the changes.

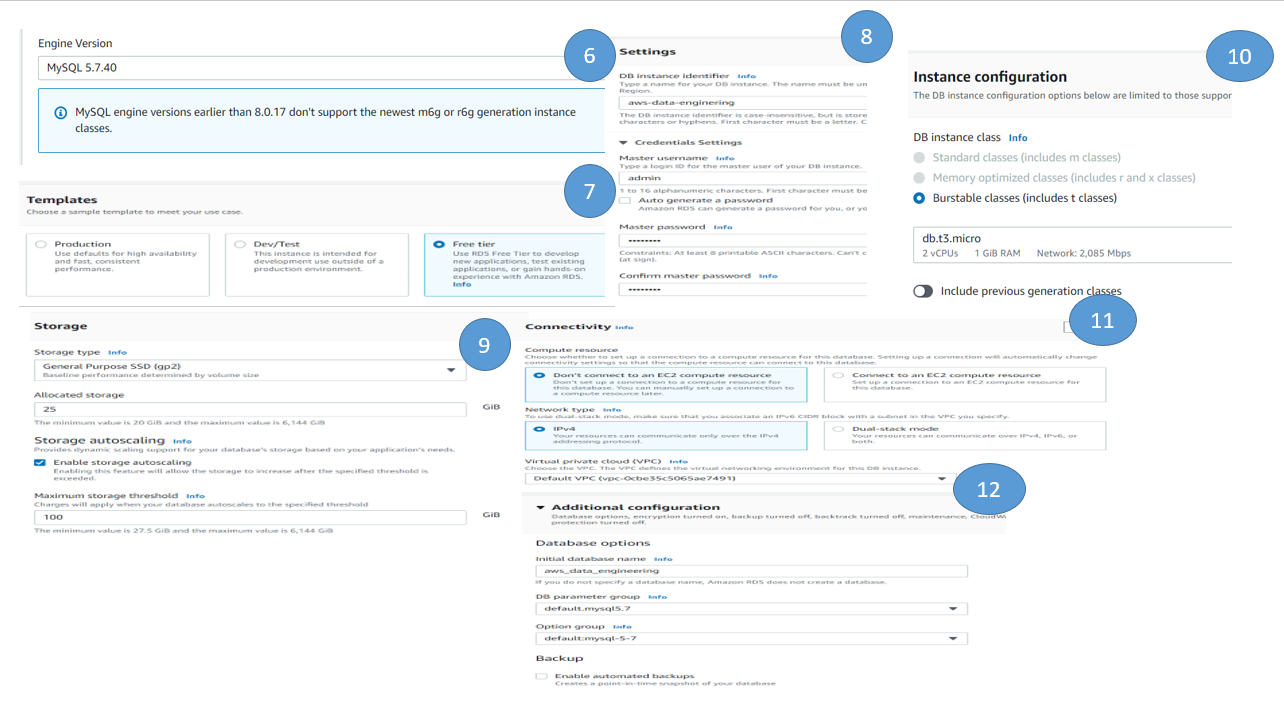


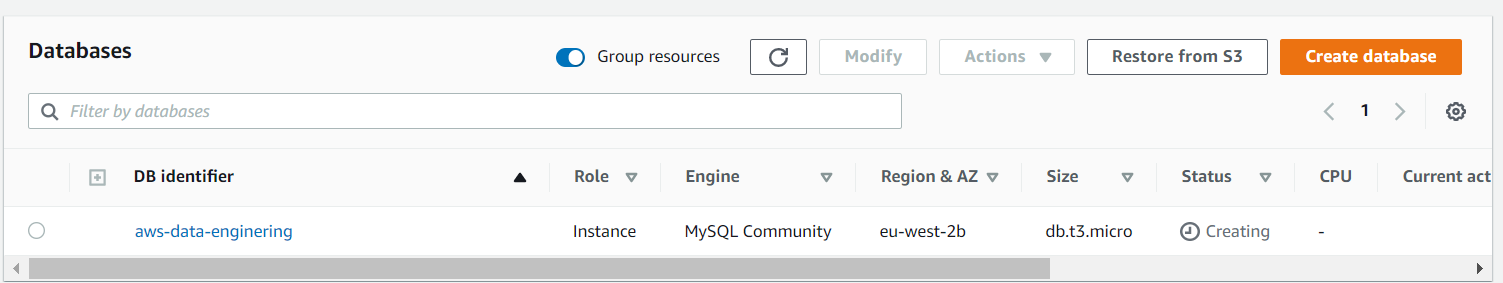
Step -6: To load the data in RDS, let’s create RDS mysql database and set up the database.

1. Search RDS in the aws console and click RDS
2. Click the database on the left hand side menu of RDS console.
3. Click on the create database to create a new database.
4. Choose create database option, select standard create as we used it for demo purpose.
5. Select database engine as mysql.



1. Select the Engine version for the database. Here we selected engine version 5.7.40 and we can easily connect from browser to write some sql.
2. Under Template, select the free tier (as we are considering this as demo, we selected the free tier but prod or dev/test, select the appropriate template)
3. Under setting, provide the require information, set your master database user name and password.
4. User can update the instance configuration as per the load requirements. We used the default configuration.
5. Under storage, select the appropriate storage requirement, since we are doing a demo, we reduced the storage to 25GB.
6. Under connectivity, select the appropriate setting. Since we are going to connect via lambda function, no change is required
7. Under additional configuration, give the initial database name so that when RDS is create, it create a default database by the name provided by user and click on the create database option to create the database.

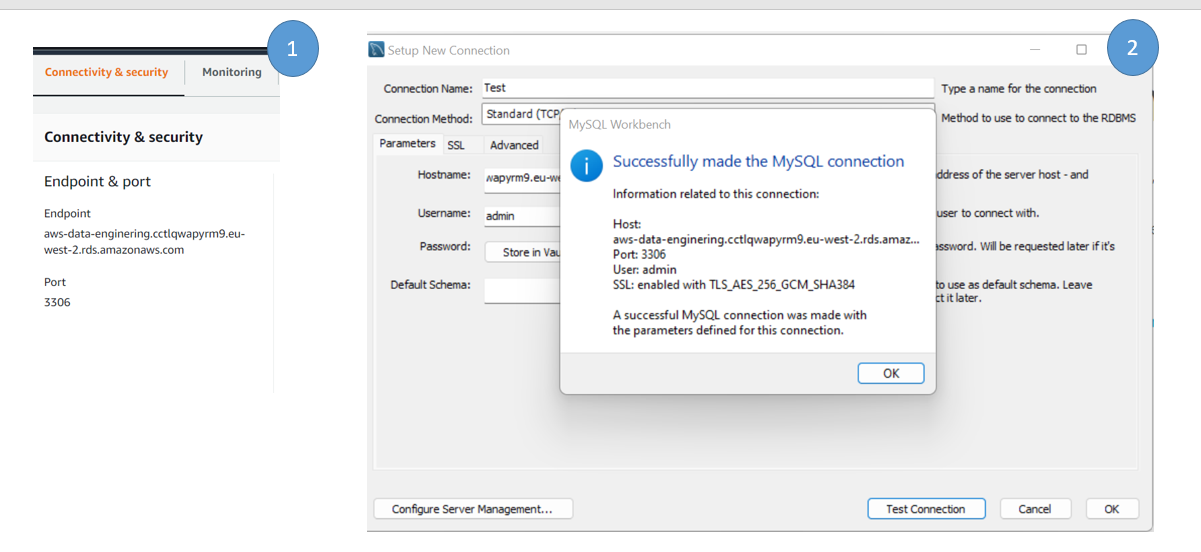




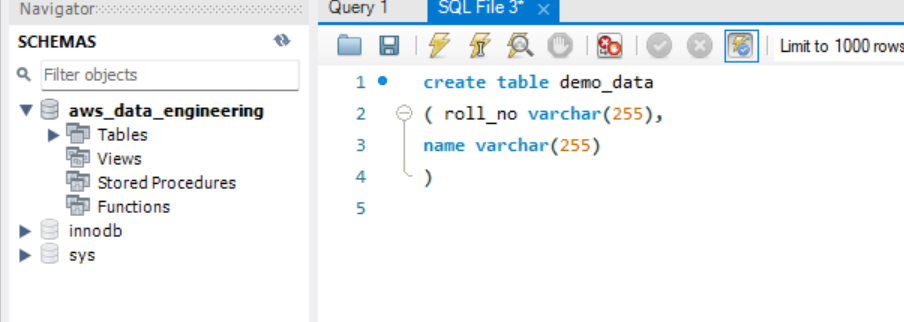
Step -7: Once the database is created, let us create a table where we are going to load the data. To login the database, we installed the mysql workbench client in our local machine ( to connect RDS, ensure you have opened the port 3306 from your IP on the inbound rule of the security group attached the RDS).

Mysql workbench download link - https://dev.mysql.com/downloads/workbench/

1. To connect RDS, get the Endpoint and port details. Click the aws-data-engineering database ( in RDS), under Connectivity and security table, get the end point details.
2. Open Mysql workbench, provide the end point details and give the admin user (which is configured during rds set up). It is recommended to create a new user like developer and use the developer user for any activity in RDS. For this demo, we used admin user id for all activities.



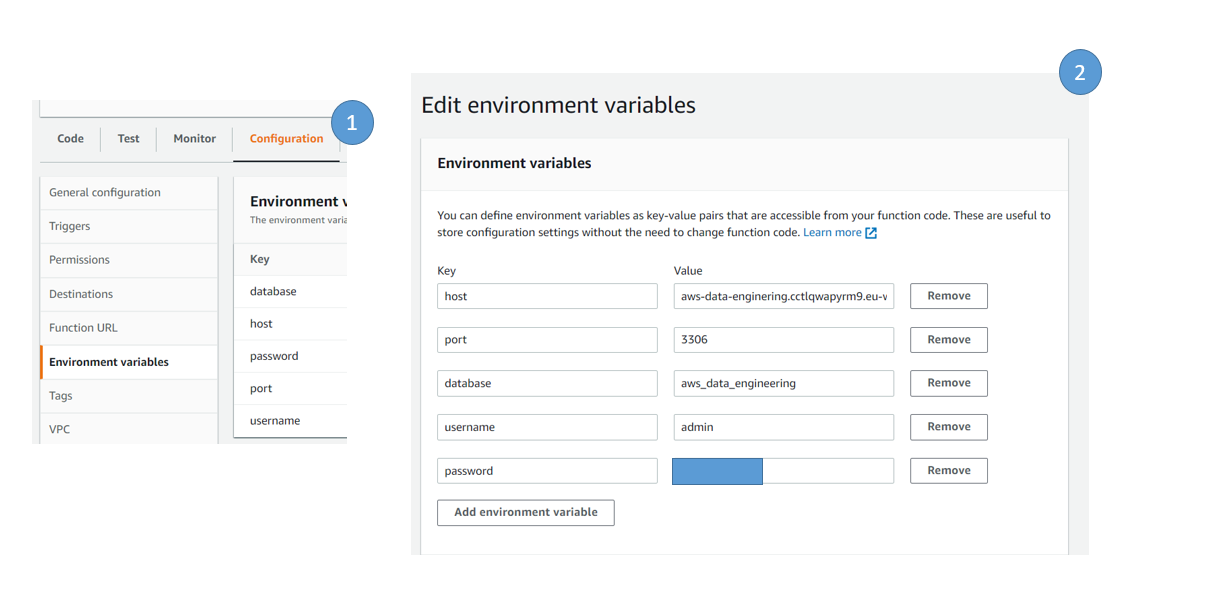
Step -8: Create a table called ***demo\_data*** under aws\_data\_engineering database.



Wonderful, if you are completed all steps so far well done. We set up a S3 bucket, lambda function, RDS, and create a table as well. The S3 bucket, we already set up the event trigger. Now we need to upload our file and see whether the data is loaded in RDS or not.

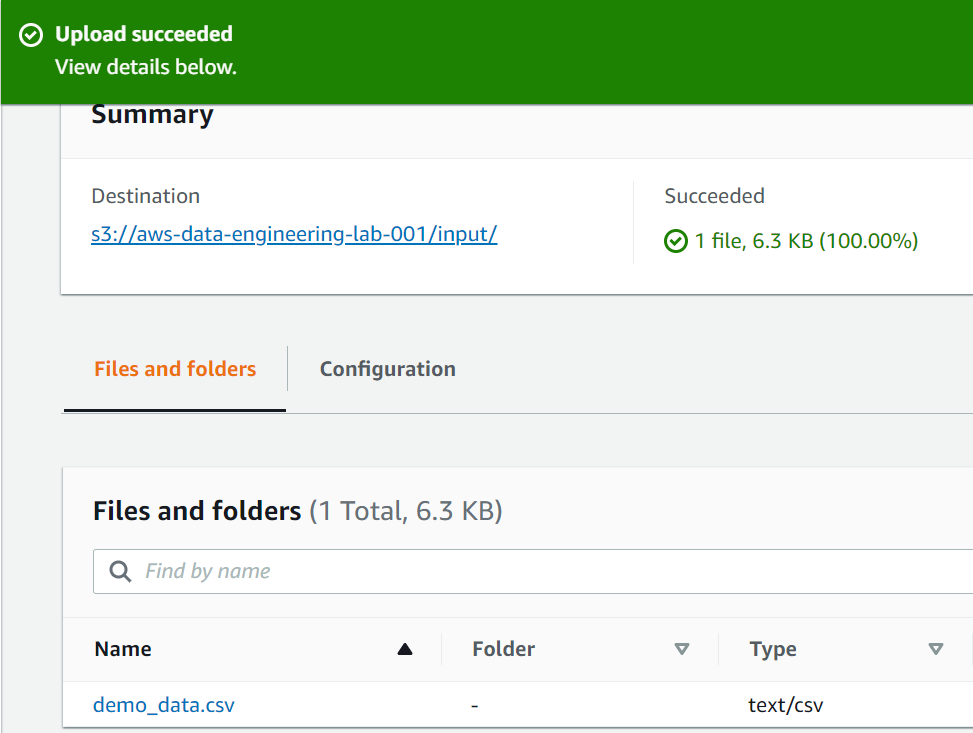
Step -9: For lambda function to connect RDS, we need to create couple of environment variables to store host, port, database, user name and password. To do the same Go to the lambda function,

1. Open the function, click on configuration tab, environment variables.
2. Add the below environment variables.

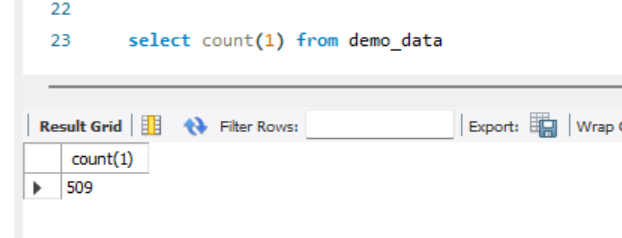


1. To connect RDS my sql from lambda function, an additional python package is require. Here we used mysql.connector package. By default this package is not available in lambda environment. Either you can create a lambda layer for this package to use. In this case, we zipped both lambda function and python package ( mysql) locally and upload it in lambda function.

Step -10: We are almost done, to test whether our program is working or not, upload a file in the S3 bucket and require folder, the event will trigger the lambda function automatically.



Step -11: We can check the lambda function execution status on cloud watch / check the record count in the table whether the data is loaded or not.



Note : This is a simple demo, however we can follow this approach and develop complex pipeline and load the data to RDS via lambda function.